Reducing hit time: small & simple caches

- Cache access time limits the clock cycle rate on many systems
  - ⇒ Cache design affects more than average memory access time - it affects everything
- Small & simple caches reduce hit time
  - – Less hardware to implement a cache => shorter critical path through the hardware
  - – Direct-mapped is faster than set-associative for both reads and writes: tag
    - There is only one block for each index
    - If tag check fails, the block is wrong and a (long) cache miss must be processed
  - – Fitting the cache on the chip with the CPU is also very important for fast access times
- Fast clock cycle time encourages small direct-mapped caches

---

Avoid address translation during indexing

- The CPU uses virtual addresses that must be mapped to a physical address
  - – The cache may either use virtual or physical addresses
- Cache indexed by virtual addresses => virtual cache
- Cache indexed by physical address => physical cache
- A virtual cache reduces hit time
  - – Translation from a virtual address to a physical address is not necessary on hits
  - – Address translation can be done in parallel with cache access => penalties for misses are reduced as well
- So why are they used so infrequently?

Issues with virtual caches

- Process switches require cache purging
  - Different processes share the same virtual addresses even though they map to different physical addresses
  - When a process is swapped out, the cache must be purged of all entries to make sure that the new process gets the correct data
- One solution: PID tags
  - Increase the width of the cache address tags to include a process ID (instead of purging the cache)
  - The current process PID is specified by a register
  - If the PID does not match, it is not a hit even if the address matches

---

Virtual caches & aliasing

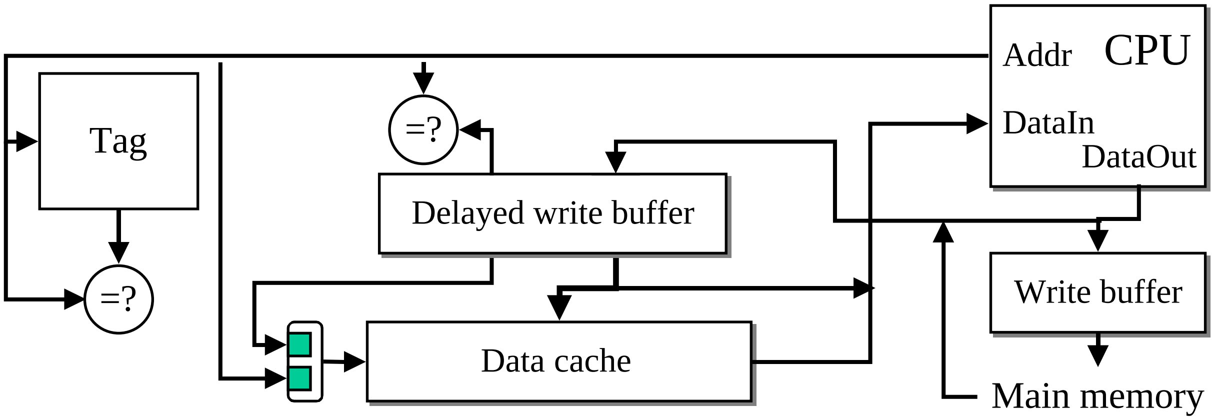
- Problem: two different virtual addresses may have the same physical address (even for a single process)
  - This may result in two copies of the same block in the cache!
  - The aliasing problem must be handled correctly
- Anti-aliasing hardware: guarantees every cache block a unique physical address
  - Every virtual address maps to the same location in the cache
  - This solution can be slow and difficult to implement in hardware
- Page coloring: software technique that forces aliases to share some address bits
  - The virtual address and physical address match over these k bits
  - A direct-mapped cache that is 2^k bytes or smaller can never have duplicate physical addresses for blocks

Reducing access time in virtual caches

- Use the page offset to index the cache: get the best of both virtual & physical caches
  - Overlap the virtual address translation process with the time required to read the tags
  - Page offset is unaffected by address translation
  - However, this restriction forces the cache size to be smaller than the page size because the index comes from the "physical" portion of the virtual address (the page offset)
- Basic operation
  - Send the page offset to the cache
  - At the same time, translate the virtual -> physical page number
  - Check the tag from cache against the physical address obtained by virtual -> physical translation
- High associativity allows for larger cache sizes

Reducing hit time with pipelined writes

- Write hits take longer than read hits because tag checking is required before the data is written
- One solution is to pipeline the writes (as in the Alpha AXP 21064)
  - The second stage of the write (cache is updated with new data) occurs during the first stage of the next write
  - Allows tag checking and data writing to occur simultaneously



Cache optimization techniques: summary

	Miss rate	Miss penalty	Hit time	Hardware complexity
Larger block sizes	+	—		0
Higher associativity	+		—	1
Victim caches	+			2
Pseudo-associativity	+			2
Hardware prefetching	+			2
Compiler-controlled prefetch	+			3
Compiler optimizations	+			0
Giving read misses priority		+		1
Subblock placement		+		1
Early restart / critical word 1st		+		2
Nonblocking cache		+		3
2nd level caches		+		2
Small & simple caches	—		+	0
Avoiding address translations			+	2
Pipelining writes			+	1

Main memory

- Main memory is usually made from DRAM while caches use SRAM
  - SRAM is faster (by almost an order of magnitude)
  - However, it's also more expensive per bit
    - DRAM uses 1 transistor & 1 capacitor per bit
    - SRAM uses 6 transistors => 4x to 8x the space
- There are methods for optimizing DRAM performance
- Performance measures for DRAM include:
  - Latency: important for caches (reduces miss penalty)
  - Bandwidth
    - Important for I/O
    - Also important for cache with second-level and larger block sizes

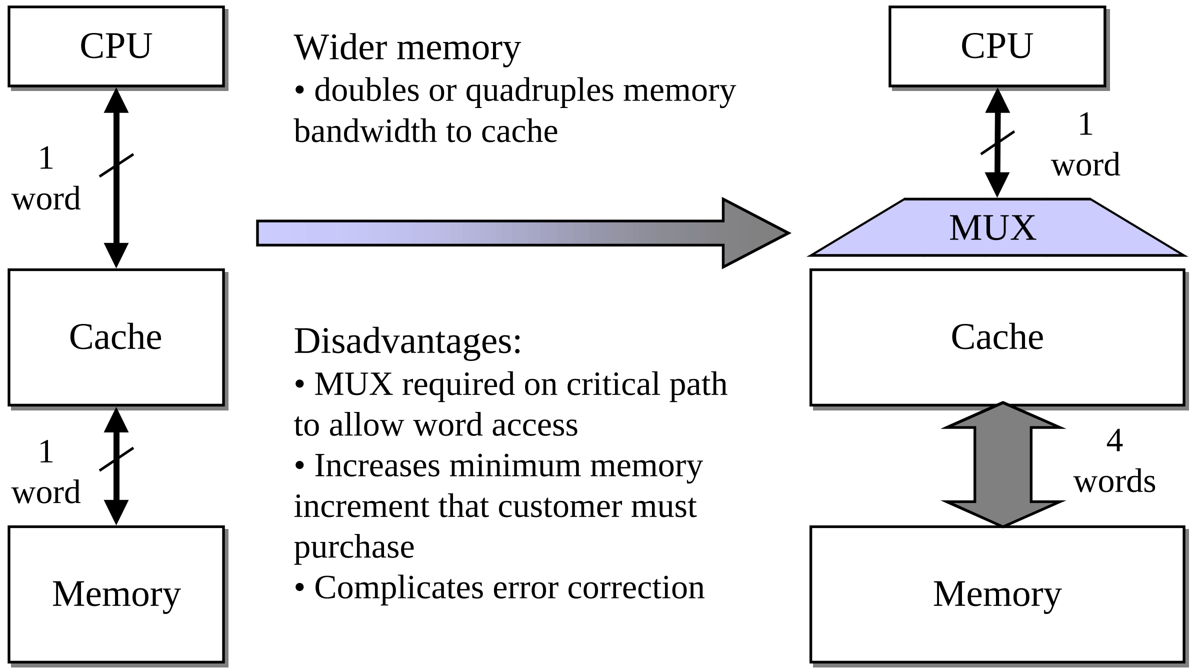
Main memory performance issues

- Latency measures include
  - – Access time: time between when a read is requested and when the desired word arrives
  - – Cycle time: the minimum time between the starts of two accesses to memory
    - This is at least as long as access time, and is usually longer
- DRAM refresh
  - – DRAMs must occasionally refresh their data
  - – This is done by reading all of the cells in a row and writing them back
  - – Refresh must be done every few milliseconds
    - This operation consumes less than 5% of total time
    - The low time requirement occurs because the time necessary to refresh is proportional to the **square root** of the size of the DRAM

Main memory performance

- Amdahl suggested that memory capacity should grow linearly with CPU speed
  - – Memory capacity grows **four-fold** every **three** years to supply this demand
  - – The CPU-DRAM performance gap is a problem, however, since DRAM performance improvement is only about 7% per year
  - – Cache innovations have addressed this problem to some degree
- There are innovations in main memory organizations that are more cost-effective

Wider main memory



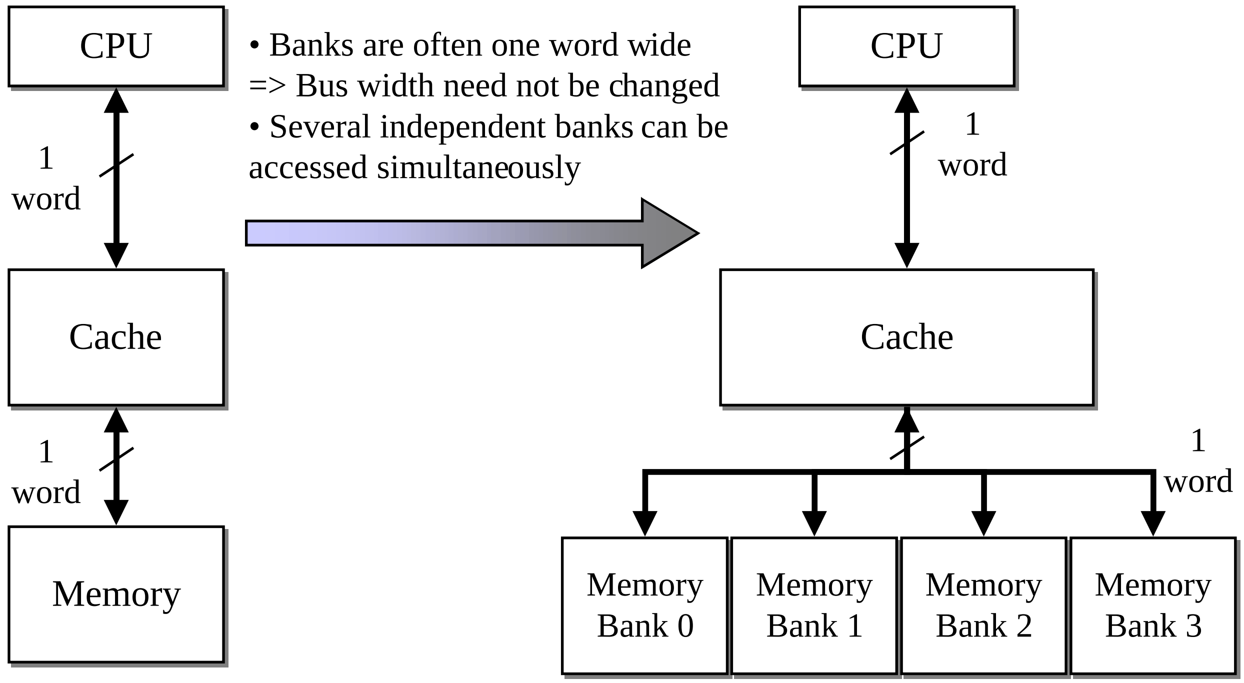
Wider memory
- doubles or quadruples memory bandwidth to cache

Disadvantages:
- MUX required on critical path to allow word access
- Increases minimum memory increment that customer must purchase
- Complicates error correction

Wider main memory

- DRAM chips are typically 1-8 bits wide
  - Any number of them can be accessed in parallel without extra delay
- By increasing the width of memory, the CPU can get more bits in a single cycle
  - This increases bandwidth between cache and memory
- Example: consider a cache with 4 word blocks
  - Main memory might require
    - 4 cycles to send the address
    - 40 cycles to access memory
    - 4 cycles to transfer over the bus
  - If the memory is only **one word** wide, a miss would require 4 x (4 + 40 + 4) = 192 cycles!
  - If the memory is enlarged to **4 words wide**, miss time is only 48 cycles

Interleaved main memory



- Banks are often one word wide
=> Bus width need not be changed
- Several independent banks can be accessed simultaneously

Interleaved main memory

- Example: fetch a block by
  - Sending 1 address
  - Waiting for a single memory cycle
  - Transferring 4 words for a total time of 4 + 40 + (4 x 4) = 60 cycles
- A little slower than wider memory (due to bus limitations)
  - Reads must transfer more words
  - Writes can be overlapped if they are addressed to different banks
- Read access optimization may be possible
  - Example: cache block size is four words => parallel access is possible
- Write-back caches make writes sequential as well as reads
- How many banks are sufficient?
  - Possible rule: ‘# of banks >= # of clocks to access a word in a bank’
  - This allows up to 1 word per clock cycle in best case

Independent memory banks

- Interleaved memory concept can be extended to remove all restrictions on memory access
  - Interleaved memory => only a single memory controller in the system
    - ⇒ Allows the interleaving of sequential access patterns
  - Address line sharing among the banks is possible in this scheme
- Instead, use multiple independent controllers
  - Example: one for I/O devices, one for cache reads and one for cache writes
  - Banks are still accessed in parallel, but now there may be multiple independent requests serviced simultaneously
- This can be particularly useful for
  - Nonblocking caches (that allow multiple outstanding read misses)
  - Multiprocessors

Avoiding memory bank conflicts

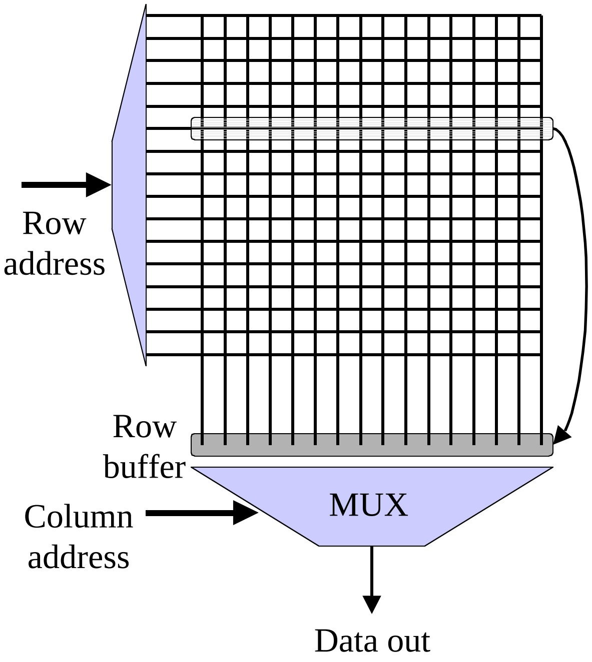
- As with caches, **programs** can be modified to improve memory performance
  - The most important principle is to keep all the banks running
  - Programs that access all banks evenly will perform best
- Problem: data memory references are **not** random and may end up going to the same bank
  - Using a **prime number** of memory banks makes this work well
  - However, using a prime number makes the division operation expensive
    Bank number = Address MOD Number of banks
    Address within bank = floor (Address / Number of banks)

Avoiding memory bank conflicts

- There are schemes distribute memory accesses to banks using
  - A prime number of banks
  - Fast modulo arithmetic
- For example, the following can be used:
  Bank number = Address MOD number of banks
  Address in bank = Address MOD number of words in bank
  - This avoids the use of an expensive 'non power of 2' division operation shown previously
  - There is a proof that guarantees that the above mapping provides a unique mapping between an address and a memory location
  - For numbers of the form 2^N-1, there is fast hardware to implement the MOD operation

Improving DRAM performance

- Previous methods work with any memory technology
- There are also techniques that take advantage of the nature of DRAMs
- DRAMs buffer a row of bits inside the DRAM for column access
  - The size of the buffer is usually the square root of the DRAM size, e.g. 16Kbits for 64MBits
- DRAMs are designed to allow multiple accesses to this buffer, **eliminating** the row access time



DRAM-specific techniques

- Nibble mode
  - The DRAM can supply three extra bits from locations sequential to the one just accessed, once after each RAS (Row Access Strobe)
- Page mode
  - The DRAM can act as an SRAM once a row has been selected
  - For example, random bits from the row can be selected by changing just the column address
  - This can occur until the next RAS or refresh
- Static column mode (Extended Data Out [EDO] RAM)
  - Very similar to page mode
  - No need to toggle (clock) the column access strobe line every time the column address changes
- These optimizations can improve bandwidth by a factor of 4!

DRAM-specific techniques

- Synchronous DRAM (SDRAM)
  - The clock is supplied to the RAM chip, and all signals are synchronized to it
  - This allows the RAM to run at higher speeds
  - Similarly, sequential data can be retrieved faster, at the rate of one bit per clock cycle (similar to page mode)
- VRAM
  - Video RAM is used to drive displays
  - Read or written using a normal interface
  - Read via special interface that outputs rows one bit at a time (good for video displays)
- Modern DRAM chips often output multiple bits at a time (4-8 bits per address)